Cairo University
Faculty of Engineering
Department of Computer Engineering

Inferex

A Graduation Project Report Submitted

to

Faculty of Engineering, Cairo University

in Partial Fulfillment of the requirements of the degree

of

Bachelor of Science in Computer Engineering.

Presented by
Mohamed Maher

Supervised by
Dr. Sandra Wahid

July 2025

Abstract

Inferex is an intelligent desktop application designed to help users recall and retrieve information they have previously seen on their computers, whether from a video, document, or website, without the need to manually save or organize it. In today's digital world, users interact with large volumes of information every day, yet remembering specific details from past activities remains a constant challenge. Most AI assistants currently available require users to manually provide the context of the question in order to receive an accurate and personalized answer, which limits their convenience and effectiveness. Inferex solves this problem by silently and privately building a local context from the user's screen activity, enabling it not only to retrieve previously seen content but also to answer related questions intelligently without requiring additional input from the user. The system operates entirely offline, ensuring complete privacy by keeping all data on the device. It captures screenshots periodically, segments them into meaningful parts, and stores them in a vector database for semantic retrieval. Lightweight algorithms and compressed models are used to guarantee smooth performance even on personal devices with limited hardware resources. Inferex offers a private, context-aware, and searchable memory of the user's digital interactions, making it a powerful tool for productivity, learning, and everyday information recall.

الملخص

إنفريكس (Inferex) هو تطبيق ذكي لسطح المكتب صُمم لمساعدة المستخدمين على تذكّر واسترجاع المعلومات التي شاهدوها سابقًا على أجهزتهم، سواء كانت من فيديو، مستند، أو موقع إلكتروني، دون الحاجة إلى حفظها أو تنظيمها يدويًا. في عالمنا الرقمي اليوم، يتفاعل المستخدمون يوميًا مع كميات ضخمة من المعلومات، ومع ذلك فإن تذكّر التفاصيل الدقيقة من الأنشطة السابقة لا يزال تحديًا مستمرًا. تعتمد معظم أدوات الذكاء الاصطناعي المتاحة حاليًا على أن يقوم المستخدم بتوفير سياق السؤال يدويًا للحصول على إجابة دقيقة وشخصية، مما يقلل من كفاءتها ويجعل استخدامها أقل سهولة. يحلّ إنفريكس هذه المشكلة من خلال إنشاء سياق محلي تلقائي وخاص بناءً على نشاط الشاشة لدى المستخدم، مما يمكّنه ليس فقط من استرجاع المحتوى الذي تم عرضه سابقًا، بل أيضًا من الإجابة على الأسئلة المتعلقة به بذكاء ودون الحاجة إلى إدخال أي معلومات إضافية. يعمل النظام بشكل كامل دون اتصال بالإنترنت، مما يضمن خصوصية المستخدم من خلال إبقاء جميع البيانات محفوظة على الجهاز فقط. يلتقط التطبيق لقطات شاشة بشكل دوري، ويقوم بتقسيمها إلى أجزاء ذات معنى، ثم يخزنها في قاعدة بيانات يمكن البحث فيها بطريقة دلالية. كما تم استخدام خوارزميات خفيفة ونماذج مضغوطة لضمان أداء سلس حتى على الأجهزة الشخصية ذات الموارد المحدودة. يقدم إنفريكس ذاكرة رقمية خاصة، مدركة للسياق، وقابلة للبحث، مما يجعله أداة قوية لزيادة الإنتاجية، وتسهيل التعلّم، واسترجاع المعلومات في الحياة اليومية.

ACKNOWLEDGMENT

I want to thank **Dr. Sandra Wahid** for her clear guidance, helpful feedback on each design version, and timely assistance whenever challenges arose. Her steady support was vital to this project's success.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
ASR	Automatic Speech Recognition
API	Application Programming Interface
CV	Computer Vision
CLAHE	Contrast Limited Adaptive Histogram Equalization
DL	Deep Learning
FFmpeg	Fast Forward Moving Picture Experts Group (Multimedia Framework)
GPU	Graphics Processing Unit
IoU	Intersection over Union
JSON	JavaScript Object Notation
mAP	Mean Average Precision
NMS	Non-Maximum Suppression
OCR	Optical Character Recognition
R-CNN	Region-based Convolutional Neural Network
ReLU	Rectified Linear Unit
RoI	Region of Interest
RPN	Region Proposal Network
UI	User Interface
URL	Uniform Resource Locator
WAV	Waveform Audio File Format

Contacts

Team Members

Name	Email	Phone Number
Mohamed Maher	mohamed.02maher@gmail.com	01121522227

Supervisor

Name	Email	Number
Sandra Wahid	sandrawahid@hotmail.com	-

This page is left intentionally empty

Chapter 1: Introduction

In Inferex, I contributed to four core modules: I developed the screen-segmentation pipeline using both SqueezeDet (with a custom loss function) and Faster R-CNN (ResNet-50 backbone), implemented a classical computer-vision model for grid extraction, and built the video-annotation system, including data labeling and front-end components.

1.1. Document Organization

This section outlines the structure of the report and provides a brief overview of the upcoming chapters. Chapter 2 presents a detailed explanation of my contributions to the modules I worked on. Chapter 3 discusses the methods used to test and evaluate these contributions. Chapter 4 offers a brief conclusion, summarizing the experience I gained and the challenges I encountered throughout the project.

Chapter 2: System Design and Architecture

In this chapter, I provide a detailed explanation of my contributions to the project.

2.3. Screen Segmentation Module:

This module is designed to segment each screenshot into grid, image, and text regions while remaining efficient on standard user hardware, even under concurrent processing loads. To tackle this challenge, our initial implementation focused on lightweight model design, which led us to adopt the ScreenSeg [9] toolkit. ScreenSeg consists of three specialized modules, with **SqueezeDet** employed specifically to detect and classify text and image regions.

SqueezeDet:

2.3.1. Functional Description

The **SqueezeDet** model detects and classifies regions within a screenshot as either *text* or *image*. It uses a lightweight, single-stage detection approach, taking a preprocessed screenshot as input and producing bounding boxes, corresponding class labels, and confidence scores for each identified region

2.3.2. Modular Decomposition

1. **Model Loading**
    - Load the optimized **(≈9 MB)** SqueezeDet checkpoint into memory.
    - Initialize the network with pretrained convolutional weights and custom detection heads.
2. **Preprocessing**
    - Resize the screenshot to the network's expected input size (e.g., 768×1024).
    - Normalize pixel values using ImageNet mean and standard deviation.

3. **Forward Pass**
    - Process the image tensor through the SqueezeNet-based convolutional backbone.
    - Extract shared feature maps for objectness prediction, bounding-box regression, and classification.

4. **Anchor Matching & Inference**
    - Predict objectness scores and bounding box offsets for each anchor box
    - Apply Non-Maximum Suppression (NMS) with an IoU threshold of 0.5 to remove redundant detections
    - Transform the final predictions back to the coordinates of the original image.

5. **Postprocessing**
    - Categorize detections into *text* or *image* groups based on class confidence scores.
    - Output the results as a list of dictionaries, each structured.

2.3.3. Loss Function

Due to the absence of a publicly available implementation of SqueezeDet, we re-implemented the training loss from scratch. The total loss consists of the following components:

1. **Objectness Loss (L_{obj}):** Binary cross-entropy loss between predicted objectness scores and ground-truth anchor labels, indicating whether an anchor corresponds to an object.

2. **Bounding-Box Regression Loss (L_{reg}):** Smooth L_1 loss applied to the predicted bounding box offsets (t_x, t_y, t_w, t_h) for anchors matched to ground-truth boxes.

3. **Class Loss (L꜀□□):** Cross-entropy loss computed over the predicted class probabilities for each positively matched anchor.

The detailed loss function

$$\frac{\lambda_{bbox}}{N_{obj}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} I_{ijk} [(\delta x_{ijk} - \delta x_{ijk}^{G})^2 + (\delta y_{ijk} - \delta y_{ijk}^{G})^2 + (\delta w_{ijk} - \delta w_{ijk}^{G})^2 + (\delta h_{ijk} - \delta h_{ijk}^{G})^2] + \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} \frac{\lambda_{conf}^{+}}{N_{obj}} I_{ijk} (\gamma_{ijk} - \gamma_{ijk}^{G})^2 + \frac{\lambda_{conf}^{-}}{WHK - N_{obj}} \bar{I}_{ijk} \gamma_{ijk}^{2} + \frac{1}{N_{obj}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} \sum_{c=1}^{C} I_{ijk} l_{c}^{G} \log(p_c).$$

Figure 2.1: Detailed training Loss function

The overall training objective is:

$$L = \lambda_{\text{obj}} L_{\text{obj}} + \lambda_{\text{reg}} L_{\text{reg}} + \lambda_{\text{cls}} L_{\text{cls}}$$

Figure 2.2: Overall training Loss function

2.4. Screen Segmentation Module: Faster R-CNN (ResNet-50)

2.4.1. Functional Overview

This module leverages a two-stage detection pipeline built on **Faster R-CNN with a ResNet-50 backbone**. In the first stage, a Region Proposal Network (RPN) generates candidate regions of interest. In the second stage, these proposals are refined and classified into *text* or *image* categories with high localization accuracy.

2.4.2. Modular Breakdown

1. **Backbone Feature Extraction**
    - The input screenshot is passed through ResNet-50 up to the *conv4* block.
    - High-resolution feature maps are generated as the foundation for detection.

2. **Region Proposal Network (RPN)**
    - A lightweight sliding convolutional layer scans the feature maps to predict:
        - Objectness scores for each anchor
        - Anchor box adjustments (offsets)
    - Top-N proposals are selected based on objectness confidence.
    - Non-Maximum Suppression (NMS) is applied to eliminate overlapping proposals.

3. **RoI Pooling & Detection Head**
    - Each region proposal is processed using **RoIAlign** to extract a fixed-size feature representation.
    - Features are passed through two fully connected layers (the detection head) to output:
        - Class probabilities (background, text, image)
        - Bounding box regressions (deltas) for each class.

4. **Inference & NMS**
    - Proposals classified as *background* are discarded.
    - Class-specific NMS (IoU > 0.5) is applied to the remaining detections.
    - The final output is formatted as a list of dictionaries:

2.4.3. Loss Function

The multi-task loss sums RPN and Fast R-CNN components:

$$L_{\text{RPN}}(\{p_i\}, \{t_i\}) = \frac{1}{N_{\text{cls}}} \sum_i L_{\text{cls}}(p_i, p_i^*) + \frac{\lambda}{N_{\text{reg}}} \sum_i p_i^* \, L_{\text{reg}}(t_i, t_i^*)$$

Figure 2.3: RPN Loss function

$$L_{\text{RCNN}}(p, u, t_u, v) = -\log p_u + 1_{u \geq 1} L_{\text{reg}}(t_u, v)$$

Figure 2.4: Classification and bounding box regression loss

2.5. Document Data Annotation Module (DocLayOut Dataset)

2.5.1. Functional Description

This module is responsible for generating high-quality, layout-aware training data using the **DocLayOut** annotation schema. It involves annotating desktop screenshots to identify key layout components—such as *text blocks* and *images*—and converting them into a structured format suitable for training and evaluation.

2.5.2. Components & Workflow

- **Annotation Source**: Desktop screenshots labeled to highlight meaningful layout regions.

- **Conversion Pipeline**:Raw detection outputs (bounding boxes, classes, scores) are processed to:
    - Remove low-confidence detections.
    - Apply NMS to eliminate redundant boxes.
    - Standardize data structure for storage and further use.

- **Output Format**:The final dataset contains a list of structured layout elements, each including:
    - Bounding box coordinates [x1, y1, x2, y2]
    - Class label ("text" or "image")

2.6. Video Annotation Module (YouTube + Local)

2.6.1. Functional Description

This module extracts **timestamped transcriptions** from video content to enrich context in the annotation pipeline. It supports both YouTube-hosted videos and local video files.

2.6.2. YouTube Transcript Workflow

- **Video ID Extraction:** Parses the YouTube URL to extract the video ID.
- **Transcript Fetching:** Uses YouTubeTranscriptApi to retrieve auto-generated subtitles (with language fallback).
- **Timestamp Formatting:** Converts start and end times into readable format (MM:SS or HH:MM:SS).
- **Output Saving:** Exports the full transcription to JSON for later use.

2.6.3. Local Video Workflow

For offline videos, the following steps are performed:

- **Audio Extraction:** Converts video to mono WAV format (16kHz) using FFmpeg for optimal Whisper input.
- **Model Loading:** Loads the Whisper ASR model ("tiny") with caching for performance.
- **Transcription:**
    - Transcribes audio using the Whisper model.
    - Segments are timestamped (in seconds) and formatted as [start_time - end_time]: text
- **Output Saving:** Exports the full transcription to JSON

2.7. Grid Extraction Module (Classical CV)

2.7.1. Functional Description

This module detects **grid-like structures** in desktop screenshots using classical computer vision techniques. Grid detection enhances layout understanding and helps correct or enrich model-based segmentation by refining spatial accuracy and enabling hierarchical organization.

2.7.2. Pipeline Stages

1. Preprocessing

The module dynamically adapts its preprocessing strategy based on image brightness:

- **Light Images:**
  - Apply median blur, Laplacian edge detection, and binarization.
  - Morphological dilation and erosion to connect structural components.

- **Dark Images:**
  - Use CLAHE (Contrast Limited Adaptive Histogram Equalization) to enhance contrast.
  - Combine blurred, CLAHE-enhanced, and Laplacian features into a weighted grayscale image.
  - Threshold, then apply morphological operations on both the image and its inverse.

2. Contour Detection

Contours are extracted from the preprocessed binary images. Candidate rectangles are selected based on:

- Minimum area threshold
- Aspect ratio constraints
- Morphological characteristics

3. Block Merging

Overlapping and intersecting bounding boxes are recursively merged into unified segments using geometric union operations. This reduces noise and consolidates fragmented grid structures.

4. Size Filtering

Large blocks occupying a disproportionate area (e.g., >67%) are discarded to maintain precision.

2.7.3. Output

- A list of bounding boxes representing the detected grid structures.
- These are used to:
    - Enhance model prediction accuracy through spatial rectification.
    - Provide hierarchical segmentation for complex UI layouts.

2.7.4. Postprocessing Usage

The detected grid blocks are further utilized in **two major ways**:

1. Block Rectification
If a grid block spatially overlaps with exactly one previously detected block from the screenshot segmentation model, it is used to **refine and correct** that block's position. Grid-derived edges are often more precise due to strong line detection, helping eliminate spatial deviation in bounding boxes.

2. Block Hierarchical Structure
If a grid block overlaps with **multiple previously detected blocks**, it is treated as a **layout segment**, and the overlapping blocks are considered its **sub-segments**. This allows the construction of a **hierarchical layout structure**, representing parent-child relationships among UI components.

2.8. Frontend Interface (Next.js + Electron)

2.8.1. Functional Description

The frontend facilitates user interaction by allowing them to ask questions and receive relevant responses. It also includes settings for a more convenient environment, packaged as a desktop application to ensure full offline functionality and local file access.

2.8.2. Technology Stack

- **Next.js Web App**
    - Built using React, Tailwind CSS, and TypeScript.
- **Electron Desktop App**
    - Wraps the Next.js server in a native application.
    - Enables access to local file systems for reading and saving annotations.

2.8.3. Key Features

- **Ask Questions** – Users can input natural-language questions through a user-friendly interface.
- **Receive Answers** – Get responses from both a large language model and an extractive question answering system.
- **Customizable Settings** – Options to enable or disable screenshot capture and data erasure for a more controlled user experience..

Chapter 3: System Testing and Verification

In this chapter, I will demonstrate how I tested and verified the correctness and effectiveness of my contribution.

3.1. Testing Setup

The testing was conducted on a laptop with the following specifications:

- CPU: AMD Ryzen 7 5800H
- RAM: 16 GB
- GPU: NVIDIA RTX 3060
- OS: Windows 11
- Python version: 3.10

3.2. Testing Plan and Strategy

For the AI-based modules, the strategy involves splitting the data into training and validation sets, monitoring the loss on both, and selecting the model version that achieves low validation loss without signs of overfitting.

3.2.1. Module Testing

3.2.1.1. Screen Segmentation (SqueezeDet and Faster R-CNN):

We evaluated both the SqueezeDet and Faster R-CNN modules under comparable conditions, using **Mean Average Precision (mAP)** with an **IoU threshold of 0.5** as the primary metric. This threshold reflects a strict requirement for spatial alignment—any predicted contour with less than 50% overlap with the ground truth is considered significantly inaccurate.

For **SqueezeDet**, we trained on **70%** of the dataset and tested on the remaining **30%**.

For **Faster R-CNN**, the dataset was split into **80% for training**, **10% for validation**, and **10% for testing**.

In both cases, we used a dataset of **25,000 annotated screenshots**, ensuring a consistent evaluation protocol across models.

Training loss	result
Squeezedet	**Text only** → 4.135
FasterRCNN ResNet 50	0.3028

Table 3.1: Training loss for SqueezeDet & FasterRcnn

Validation mAP	result
Squeezedet	The best epoch is 100 Both classes → 38.3% **Text only** → 53.8%
FasterRCNN ResNet 50	The best epoch is 7 Image → 64.6% , Text → 74.3% mAP → 69.5%

Table 3.2: mAP Validation for SqueezeDet & FasterRcnn

test mAP	result
FasterRCNN ResNet 50	Image → 55.2% , Text → 73.2% mAP → 64.2%

Table 3.3: mAP Testing for FasterRcnn

3.2.1.2. GRID Identification:

Testing Challenge

Due to the absence of a ground truth dataset for grid structures in desktop screenshots, direct accuracy measurement (e.g., precision, recall, F1-score) is not feasible. Instead, the module is validated through a combination of **visual inspection** and **interactive parameter tuning**.

1. Qualitative Visual Inspection

The primary evaluation involves overlaying predicted grid bounding boxes on the original screenshots and manually reviewing their correctness. Key goals of the inspection include:

- **Correct Identification** of structured layout regions (e.g., tables, lists, carousels).
- **Minimal False Positives** on irregular or non-repetitive UI content.
- **Proper Merging** of overlapping or fragmented blocks into coherent and complete segments.

2. Interactive Grid Detection Testing Tool

To support visual evaluation and parameter tuning, a **custom OpenCV-based testing interface** was developed. The tool allows real-time adjustment of preprocessing and detection parameters through GUI trackbars, enabling live feedback on detection performance. This makes it easier to calibrate for diverse UI layouts and lighting conditions.

3.2.1.3. Activity Finder:

For YouTube Video:

- Tested valid, invalid, and malformed YouTube URLs for proper video ID extraction.
- Confirmed handling of missing, restricted, or unsupported videos.
- Verified transcript retrieval in the requested or fallback language.
- Checked timestamp formatting: [MM: SS - MM: SS] and [HH:MM: SS - HH:MM: SS].
- Validated JSON output structure with video URL and transcript.

For Local Video:

- Tested valid and invalid file paths, unsupported formats, and corrupted files.
- Verified FFmpeg conversion to mono 16kHz WAV.
- Confirmed cleanup of temporary audio files.
- Checked Whisper transcription with different model sizes and device settings.
- Validated [start-end] timestamps and output formatting.
- Confirmed correct JSON structure and safe filename generation.

Error Handling

- Verified error catching for FFmpeg, Whisper, and YouTube transcript API.
- Confirmed graceful handling of missing transcripts, broken links, and failed conversions.

3.3. Testing Schedule

Each component was tested immediately upon completion to verify its functionality and correctness before moving on to the next stage.

Chapter 4: Conclusions and Faced Challenges

4.1. Faced Challenges

A major obstacle was the absence of publicly available, fully annotated datasets for desktop UI screenshots. While we found partial datasets, they often lacked consistent bounding box annotations or had labels that did not align with our defined classes (e.g., text vs. image). This made supervised training difficult and forced us to rely on a combination of semi-automatic annotation, manual verification, and preprocessing pipelines to create usable ground truth for training and evaluation. The inconsistency and sparsity of labels also complicated the benchmarking of model performance.

Adapting the SqueezeDet architecture for desktop screenshots required multiple trials. The model exhibited **low precision**, especially for image regions, due to the domain gap between the original dataset (used in automotive contexts) and our target UI layouts. Despite extensive tuning, the model underperformed compared to two-stage detectors. Consequently, we prioritized the **Faster R-CNN (ResNet-50)** model for final segmentation tasks due to its superior accuracy and robustness.

Testing the grid identification module posed a unique challenge: there is **no standard ground truth dataset** for grid structures in desktop UI screenshots. As a result, we could not use standard quantitative metrics like precision or recall. Instead, we relied heavily on **visual inspection**, custom interactive tuning tools, and manual validation to ensure that the predicted grid blocks aligned with perceived UI structure. This made performance evaluation more subjective and labor-intensive.

Extracting timestamped transcriptions from videos was **extremely time-consuming** during annotation. To speed up the process, I adopted **pragmatic shortcuts**: for **YouTube videos**, I used **existing auto-generated subtitles** via API instead of downloading and transcribing audio manually. For **local videos**, I chose the **“tiny” variant of the Whisper model**, which offered fast transcription with acceptable accuracy. These optimizations significantly reduced annotation time and enabled faster integration into the system.

4.2. Gained Experience

Throughout the development of this system, I acquired a wide range of technical and problem-solving skills that deepened my understanding of practical AI deployment and system engineering.

Working with both deep learning (SqueezeDet, Faster R-CNN) and classical computer vision techniques gave me hands-on experience in model selection, optimization, and trade-off analysis between speed and accuracy. I learned how to choose the right tool for the job and adapt preexisting models to new domains through architecture tuning and retraining.

Due to the absence of perfect training data, I learned how to construct semi-automated annotation pipelines, convert raw model outputs into clean formats, and apply pre-/postprocessing strategies (e.g., NMS, thresholding) to produce usable datasets. This was critical in maintaining training quality while reducing manual effort.

The grid detection module taught me how to verify system correctness through qualitative evaluation in the absence of conventional metrics. Designing a parameter tuning interface using OpenCV and relying on visual feedback gave me a strong understanding of how to evaluate model output when quantitative benchmarks are not available.

In the activity finder module, I learned to make design choices that balance quality with processing speed. This included leveraging auto-generated YouTube subtitles and selecting lighter ASR models (like Whisper-tiny) to optimize time during annotation. These decisions helped maintain workflow scalability without sacrificing functionality.

4.3. Conclusions

This project resulted in a modular screen analysis system capable of segmenting desktop UI content, detecting layout structures, and extracting video-based activity. By combining Faster R-CNN for accurate region detection, classical CV for grid refinement, and lightweight transcription tools for video annotation, the system achieved robust performance across diverse input types.

Despite challenges like limited annotated data and a lack of ground truth for grid structures, careful tuning, visual inspection, and practical optimization strategies helped overcome these limitations. The project demonstrated the value of combining deep learning with traditional methods and offered a strong foundation for future enhancements.

References

[1] B. Wu, A. Wan, F. Iandola, P. H. Jin, and K. Keutzer, “SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving,” arXiv preprint arXiv:1612.01051, 2019.

[2] M. Goyal, D. Garg, R. S. Munjal, D. P. Mohanty, S. Moharana, and S. P. Thota, “ScreenSeg: On-Device Screenshot Layout Analysis,” arXiv preprint arXiv:2104.08052, 2021.